Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №3 по курсу «Моделирование»

Тема ОДУ второго порядка с краевыми условиями 2-го и 3-го рода

Студент Кононенко С.С.

Группа ИУ7-63Б

Оценка (баллы) ________________

Преподаватель Градов В.М.

Москва — 2021 г.

Тема работы

Программно-алгоритмическая реализация моделей на основе ОДУ второго порядка с краевыми условиями II и III рода.

Цель работы

Получение навыков разработки алгоритмов решения краевой задачи при реализации моделей, построенных на ОДУ второго порядка.

Теоретические сведения

Задана математическая модель:

$$\frac{d}{dx}(\lambda(T)\frac{dT}{dx}) - 4 \cdot k(T) \cdot n_p^2 \cdot \sigma \cdot (T^4 - T_0^4) = 0$$

Для модели заданы краевые условия:

$$\begin{cases} x = 0, -\lambda(T(0))\frac{dT}{dx} = F_0. \\ x = l, -\lambda(T(l))\frac{dT}{dx} = \alpha(T(l) - T_0) \end{cases}$$

Функции $\lambda(T)$ и $k(T)$ заданы таблицей (таблица приведена отдельно).

Заданы начальные параметры:

- $n_p = 1.4$ – коэффициент преломления;
- $l = 0.2$ см – толщина слоя;
- $T_0 = 300$K – температура окружающей среды;

- $\sigma = 5.668 \cdot 10^{-12}\ \frac{\text{Вт}}{\text{см}^2 \cdot \text{K}^4}$ – постоянная Стефана–Больцмана;
- $F_0 = 100\ \frac{\text{Вт}}{\text{см}^2}$ – поток тепла;
- $\alpha = 0.05\ \frac{\text{Вт}}{\text{см}^2 \cdot \text{K}}$ – коэффициент теплоотдачи.

Выход из итераций организовать по температуре и по балансу энергии:

$$max|\frac{y_n^s - y_n^{s-1}}{y_n^s}| <= \varepsilon_1$$

для всех $n = 0, 1, ...N.$ и

$$max|\frac{f_1^s - y_2^s}{f_1^s}| <= \varepsilon_1$$

где

$$f_1 = F_0 - \alpha(T(l) - T_0)$$

$$f_2 = 4n_p^2\sigma_0^1 k(T(x))(T^4(x) - T_0^4)dx$$

Исходный код алгоритмов

В листинге 1 представлена реализация алгоритма решения задачи. В листингах 2 – 5 приведены вспомогательные функции и главная программа.

Листинг 1 – Реализация алгоритма решения задачи

```
package emission

import (
    "fmt"
    "math"

    "gonum.org/v1/gonum/integrate"
    "gonum.org/v1/gonum/interp"
)

func SimpleIters(al, eps float64, mits int) FArr64 {
    lt := interpolate(LambdaTbl[0], LambdaTbl[1])
    kt := interpolate(KTbl[0], KTbl[1])
    its := 0
```

```go

    ts := make(FArr64, int(Params.L/Params.H)+1)
    for i := 0; i < len(ts); i++ {
        ts[i] = Params.T0
    }

    lcs := getLConds(lt, kt, ts)
    rcs := getRConds(lt, kt, ts)

    a, b, c, d := calcCoeffs(kt, lt, ts)

    tsn := runningThrough(a, b, c, d, lcs, rcs)
    for math.Abs((getF1(tsn)-getF2(kt, tsn))/getF1(tsn)) > eps && its < mits {
        ts = tsn
        lcs = getLConds(lt, kt, ts)
        rcs = getRConds(lt, kt, ts)
        a, b, c, d = calcCoeffs(kt, lt, ts)

        tsxr := runningThrough(a, b, c, d, lcs, rcs)
        for i := 0; i < len(ts); i++ {
            tsn[i] = ts[i] + Params.Alpha*(tsxr[i]-ts[i])
        }

        its++
    }

    return tsn
}

func calcCoeffs(kt, lt interp.AkimaSpline, tbl FArr64) (FArr64, FArr64, FArr64, FArr64) {
    n := int(Params.L / Params.H)
    a := make(FArr64, n-1)
    b := make(FArr64, n-1)
    c := make(FArr64, n-1)
    d := make(FArr64, n-1)

    for i := 1; i < n; i++ {
        lnm := lt.Predict(tbl[i-1])
        ln := lt.Predict(tbl[i])
        lnp := lt.Predict(tbl[i+1])

        a[i-1] = getXi(lnm, ln) / Params.H
        c[i-1] = getXi(ln, lnp) / Params.H
        b[i-1] = a[i-1] + c[i-1] + getP(tbl[i], kt)*Params.H
        d[i-1] = getF(tbl[i], kt) * Params.H
    }

    return a, b, c, d
```

```
}

func runningThrough(a, b, c, d FArr64, lcs, rcs Conds) FArr64 {
    n := len(a)

    xil := FArr64{-lcs.M / lcs.K}
    etal := FArr64{lcs.P / lcs.K}

    for i := 0; i < n; i++ {
        xil = append(xil, c[i]/(b[i]-a[i]*xil[i]))
        etal = append(etal, (d[i]+a[i]*etal[i])/(b[i]-a[i]*xil[i]))
    }

    y := FArr64{(rcs.P - rcs.K*etal[n]) / (rcs.M + rcs.K*xil[n])}
    for i := n; i > -1; i-- {
        y = append([]float64{xil[i]*y[0] + etal[i]}, y...)
    }

    return y
}

func getLConds(lt, kt interp.AkimaSpline, tbl FArr64) Conds {
    var cs Conds

    hs := Params.H * Params.H
    xif := getXi(lt.Predict(tbl[0]), lt.Predict(tbl[1]))
    pf := getXi(getP(tbl[0], kt), getP(tbl[1], kt))
    ff := getXi(getF(tbl[0], kt), getF(tbl[0], kt))

    cs.K = xif + hs/8*pf + hs/4*getP(tbl[0], kt)
    cs.M = hs/8*pf - xif
    cs.P = Params.H*Params.F0 + hs/4*(getF(tbl[0], kt)+ff)

    return cs
}

func getRConds(lt, kt interp.AkimaSpline, tbl FArr64) Conds {
    var cs Conds

    hs := Params.H * Params.H
    xif := getXi(lt.Predict(tbl[len(tbl)-1]), lt.Predict(tbl[len(tbl)-2]))
    pn := getP(tbl[len(tbl)-1], kt)
    pf := getXi(getP(tbl[len(tbl)-1], kt), getP(tbl[len(tbl)-2], kt))
    fn := getF(tbl[len(tbl)-1], kt)
    ff := getXi(getF(tbl[len(tbl)-1], kt), getF(tbl[len(tbl)-2], kt))

    cs.K = xif - hs/8*pf
    cs.M = -Params.H*Params.Alpha - xif - hs/8*pf - hs/4*pn
```

```
    cs.P = -Params.H*Params.Alpha*Params.T0 - hs/4*(fn+ff)

    return cs
}

func interpolate(xs, ys FArr64) interp.AkimaSpline {
    var as interp.AkimaSpline

    err := as.Fit(xs, ys)
    if err != nil {
        fmt.Println("Failed to initialize spline")
    }

    return as
}

func getP(x float64, kt interp.AkimaSpline) float64 {
    return 0
}

func getF(x float64, kt interp.AkimaSpline) float64 {
    return -4 * Params.Np * Params.Np * Params.Sigma * kt.Predict(x) * (math.Pow(x, 4) -
        math.Pow(Params.T0, 4))
}

func getXi(x1, x2 float64) float64 {
    return (x1 + x2) / 2
}

func getF1(tbl FArr64) float64 {
    return Params.F0 - Params.Alpha*(tbl[len(tbl)-1]-Params.T0)
}

func getF2(kt interp.AkimaSpline, tbl FArr64) float64 {
    x := make(FArr64, int((Params.L+Params.H)/Params.H))
    for i := 1; i < len(x); i++ {
        x[i] = x[i-1] + Params.H
    }
    y := make(FArr64, len(tbl))
    for i := 0; i < len(y); i++ {
        y[i] = kt.Predict(tbl[i]) * (math.Pow(tbl[i], 4) - math.Pow(Params.T0, 4))
    }
    return integrate.Simpsons(x, y)
}
```

Листинг 2 – Реализация вспомогательных типов

```
package emission

```

```
// FArr64 is used to represent []float64
type FArr64 []float64

// FMat64 is used to represent [][]float64
type FMat64 []FArr64

// Conds is used to represent emission system conditions
type Conds struct {
    K float64
    M float64
    P float64
}

// Emission is used to represent emission system parameters
type Emission struct {
    Np float64
    L float64
    T0 float64
    Tconst float64
    Sigma float64
    F0 float64
    Alpha float64
    H float64
}
```

Листинг 3 – Константы

```
package emission

var (
    LambdaTbl = FMat64{
        FArr64{300, 500, 800, 1100, 2000, 2400},
        FArr64{1.36e-2, 1.63e-2, 1.81e-2, 1.98e-2, 2.50e-2, 2.74e-2},
    }
    KTbl = FMat64{
        FArr64{293, 1278, 1528, 1677, 2000, 2400},
        FArr64{2.0e-2, 5.0e-2, 7.8e-2, 1.0e-1, 1.3e-1, 2.0e-1},
    }
    Params = Emission{1.4, 0.2, 300, 400, 5.668e-12, 100, 0.05, 1e-4}
)
```

Листинг 4 – Реализация функций отрисовки графика

```
package emission

import (
    "fmt"
    "image/color"
    "os"
```

```
    "gonum.org/v1/plot"
    "gonum.org/v1/plot/plotter"
    "gonum.org/v1/plot/vg"
)

// DrawPlot is used to draw plot with given coordinates and meta info
func DrawPlot(xs, ys FArr64, title, xl, yl, file string) {
    p := plot.New()

    p.Title.Text = title
    p.X.Label.Text = xl
    p.Y.Label.Text = yl
    p.Add(plotter.NewGrid())

    dots := convertDots(xs, ys)

    l, err := plotter.NewLine(dots)
    if err != nil {
        fmt.Println("Error:", err)
        os.Exit(1)
    }
    l.LineStyle.Width = vg.Points(1)
    l.LineStyle.Color = color.RGBA{B: 255, A: 255}

    p.Add(l)

    if err := p.Save(10*vg.Inch, 4*vg.Inch, file); err != nil {
        panic(err)
    }
}

// DrawMultiplePlot is used to draw multiple plots with given coordinates and meta info
    on one plot
func DrawMultiplePlot(xs1, ys1, xs2, ys2 FArr64, title, xl, yl, file string) {
    p := plot.New()

    p.Title.Text = title
    p.X.Label.Text = xl
    p.Y.Label.Text = yl
    p.Add(plotter.NewGrid())

    dots1 := convertDots(xs1, ys1)
    dots2 := convertDots(xs2, ys2)

    l1, err := plotter.NewLine(dots1)
    if err != nil {
        fmt.Println("Error:", err)
```

```
        os.Exit(1)
    }
    l1.LineStyle.Width = vg.Points(1)
    l1.LineStyle.Color = color.RGBA{B: 255, A: 255}

    l2, err := plotter.NewLine(dots2)
    if err != nil {
        fmt.Println("Error:", err)
        os.Exit(1)
    }
    l2.LineStyle.Width = vg.Points(1)
    l2.LineStyle.Color = color.RGBA{R: 255, A: 255}

    p.Add(l1, l2)

    if err := p.Save(10*vg.Inch, 4*vg.Inch, file); err != nil {
        panic(err)
    }
}

func convertDots(xs, ys FArr64) plotter.XYs {
    var conv plotter.XYs

    for i := 0; i < len(xs); i++ {
        d := plotter.XY{
            X: xs[i],
            Y: ys[i],
        }
        conv = append(conv, d)
    }

    return conv
}
```

Листинг 5 – Главная программа

```
package main

import (
    "lab_03/emission"
)

func main() {
    xl := make(emission.FArr64,
        int((emission.Params.L+emission.Params.H)/emission.Params.H))
    for i := 1; i < len(xl); i++ {
        xl[i] = xl[i-1] + emission.Params.H
    }
    {
```

```
        tbl := emission.SimpleIters(0.25, 1e-5, 100)
        emission.DrawPlot(xl, tbl, "T(x)", "x", "T", "data/tx.png")
    }
    {
        emission.Params = emission.Emission{1.4, 0.2, 300, 400, 5.668e-12, 100, 0.05,
            1e-4}
        emission.Params.T0 = 1200
        tbl := emission.SimpleIters(0.25, 1e-5, 100)
        emission.DrawPlot(xl, tbl, "T(x)", "x", "T", "data/tx0.png")
    }
    {
        emission.Params = emission.Emission{1.4, 0.2, 300, 400, 5.668e-12, 100, 0.05,
            1e-4}
        emission.Params.F0 = -10
        tbl := emission.SimpleIters(0.25, 1e-5, 100)
        emission.DrawPlot(xl, tbl, "T(x)", "x", "T", "data/tx1.png")
    }
    {
        emission.Params = emission.Emission{1.4, 0.2, 300, 400, 5.668e-12, 100, 0.05,
            1e-4}
        tbl1 := emission.SimpleIters(0.25, 1e-5, 100)
        emission.Params.Alpha *= 3
        tbl2 := emission.SimpleIters(0.25, 1e-5, 100)
        emission.DrawMultiplePlot(xl, tbl1, xl, tbl2, "T(x)", "x", "T", "data/tx2.png")
    }
    {
        emission.Params = emission.Emission{1.4, 0.2, 300, 400, 5.668e-12, 100, 0.05,
            1e-4}
        emission.Params.F0 = 0
        tbl := emission.SimpleIters(0.25, 1e-5, 100)
        emission.DrawPlot(xl, tbl, "T(x)", "x", "T", "data/tx3.png")
    }
}
```

Результат работы программы

Представить разностный аналог краевого условия при $x = l$ и его краткий вывод интегро-интерполяционным методом.

Для исходной системы

$$\begin{cases} \frac{d}{dx}(\lambda(T)\frac{dT}{dx}) - 4 \cdot k(T) \cdot n_p^2 \cdot \sigma \cdot (T^4 - T_0^4) = 0 \\ x = 0, -\lambda(T(0))\frac{dT}{dx} = F_0 \\ x = l, -\lambda(T(l))\frac{dT}{dx} = \alpha(T(l) - T_0) \end{cases}$$

приведем уравнение к виду:
$\frac{d}{dx}(k(u,x)\frac{du}{dx}) - p(x,u) + f(x,u) = 0$, где
$p(x,u) = 0$,
$f(x,u) = f(u) = -4 \cdot k(T) \cdot n_p^2 \cdot \sigma \cdot (T^4 - T_0^4)$.

Введем обозначение:
$F = -\lambda(x)\frac{dT}{dx}$

Рассчитаем интеграл:
$\int_{X_{N-\frac{1}{2}}}^{X_N} \frac{dF}{dx}dx + \int_{X_{N-\frac{1}{2}}}^{X_N} f(x)dx = 0$
$-(F_N - F_{N-\frac{1}{2}}) + \frac{h}{4}(f_N + f_{N-\frac{1}{2}}) = 0$

Учитывая, что
$F_N = \alpha_N(y_N - T_0)$,
$F_{N-\frac{1}{2}} = X_{N-\frac{1}{2}}\frac{y_{N-1}-y_N}{h}$

Получаем
$\alpha_N(y_N - T_0) - \frac{h}{4}(f_N + f_{N-\frac{1}{2}}) = X_{N-\frac{1}{2}}\frac{y_{N-1}-y_N}{h}$,
$-T_0\alpha_N - \frac{h}{4}(f_N + f_{N-\frac{1}{2}}) = y_{N_1}\frac{X_{N-\frac{1}{2}}}{h} + y_N(-\alpha_N - \frac{X_{N-\frac{1}{2}}}{h})$

Отсюда получаем краевое условие
$K_N y_{N-1} + M_N y_N = P_N$

График зависимости температуры $T(x)$ координаты x при заданных выше параметрах.

На рисунке 1 представлен график зависимости $T(x)$ при заданных выше параметрах.

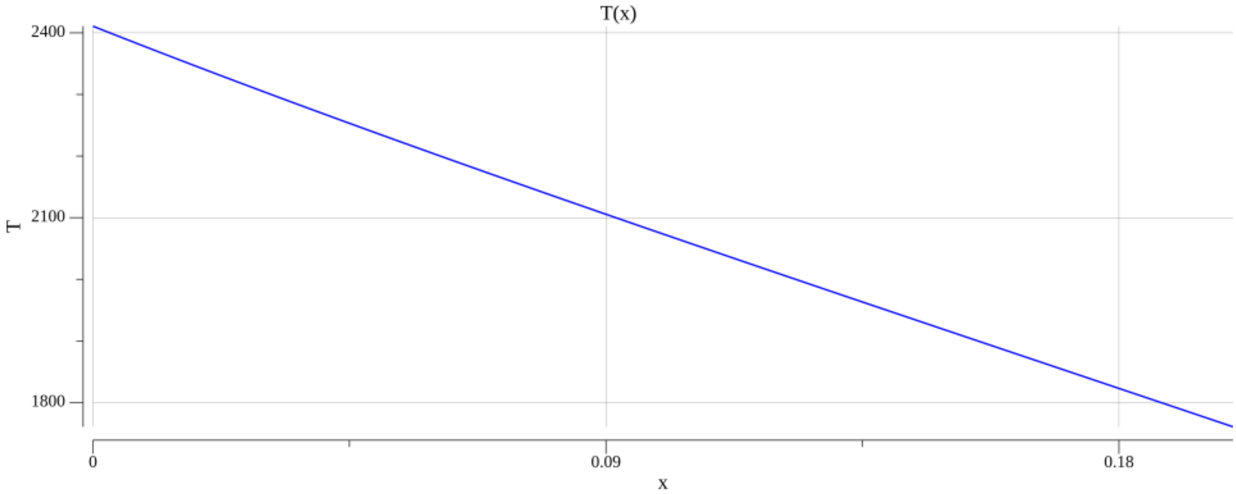

Рисунок 1 – График зависимости $T(x)$ при заданных выше параметрах

На рисунке 2 представлен график зависимости $T(x)$ при $T_0 = 1200$. Для нахождения результата понадобилось меньше итераций.

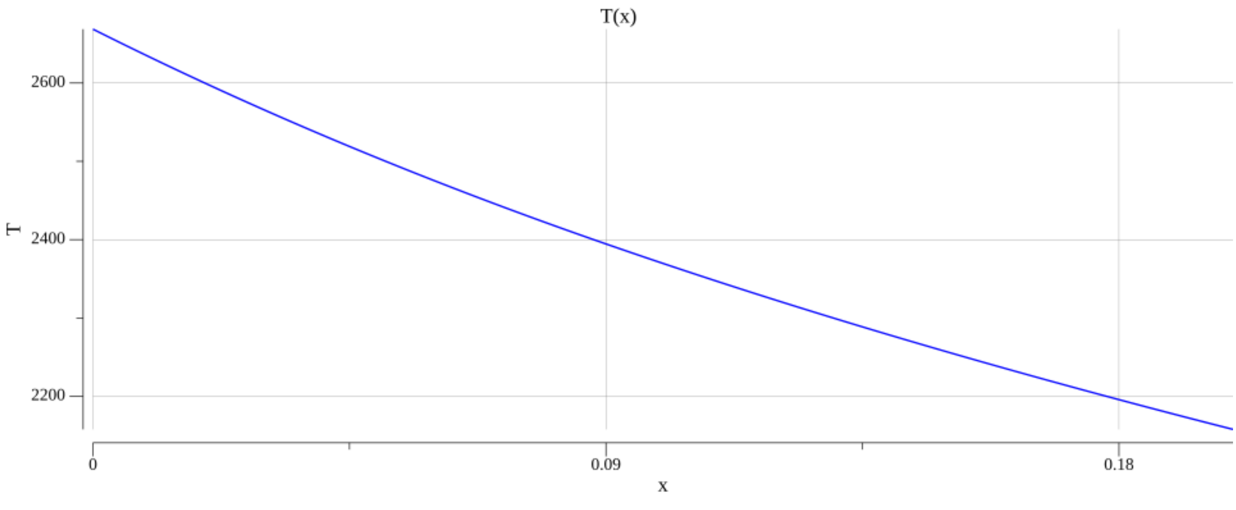

Рисунок 2 – График зависимости $T(x)$ при $T_0 = 1200$

График зависимости $T(x)$ при $F_0 = $ -10 $\frac{\text{Вт}}{\text{см}^2}$.

На рисунке 3 представлен график зависимости $T(x)$ при $F_0 = $ -10 $\frac{\text{Вт}}{\text{см}^2}$.

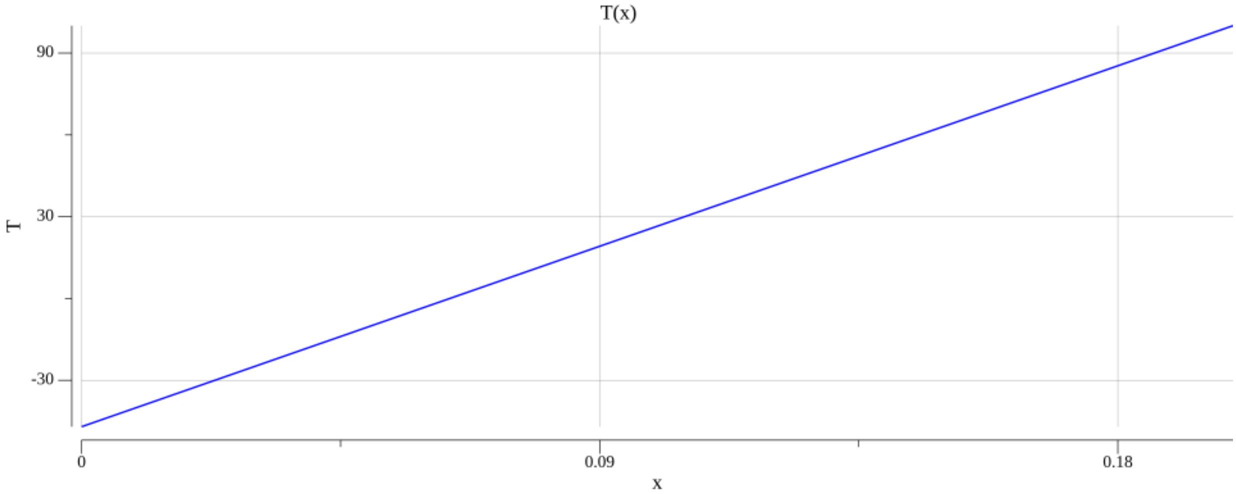

Рисунок 3 – График зависимости $T(x)$ при $F_0 = $ -10 $\frac{\text{Вт}}{\text{см}^2}$

График зависимости $T(x)$ при увеличенных значениях α (например, в 3 раза). Сравнить с п. 2.

На рисунке 4 представлен график зависимости $T(x)$ при увеличенных значениях α (красная линия – увеличенное значение, синяя – значения из п. 2.).

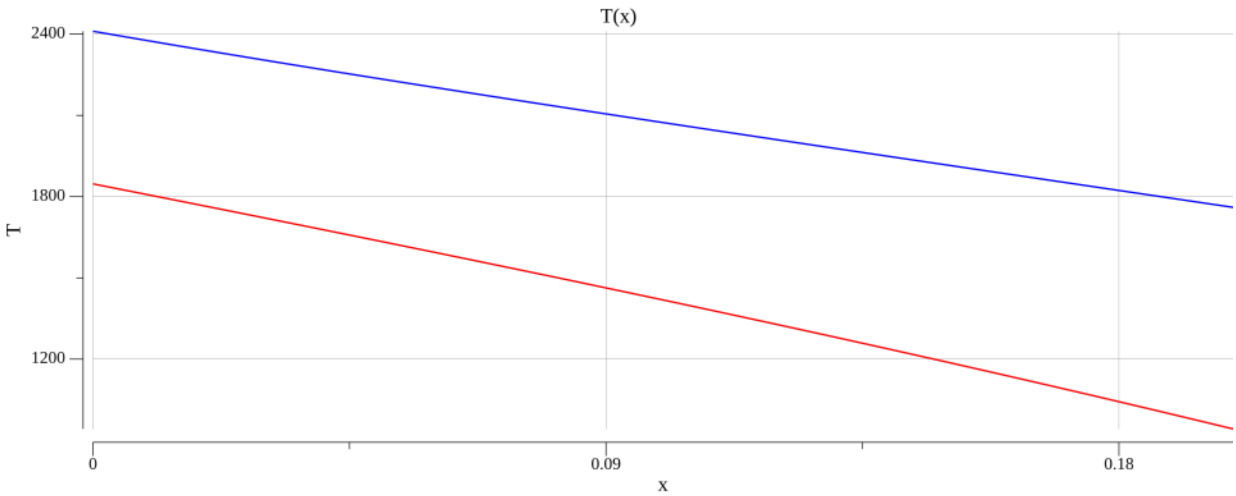

Рисунок 4 – График зависимости $T(x)$ при увеличенных значениях α

График зависимости $T(x)$ при $F_0 = 0$.

На рисунке 5 представлен график зависимости $T(x)$ при $F_0 = 0$.

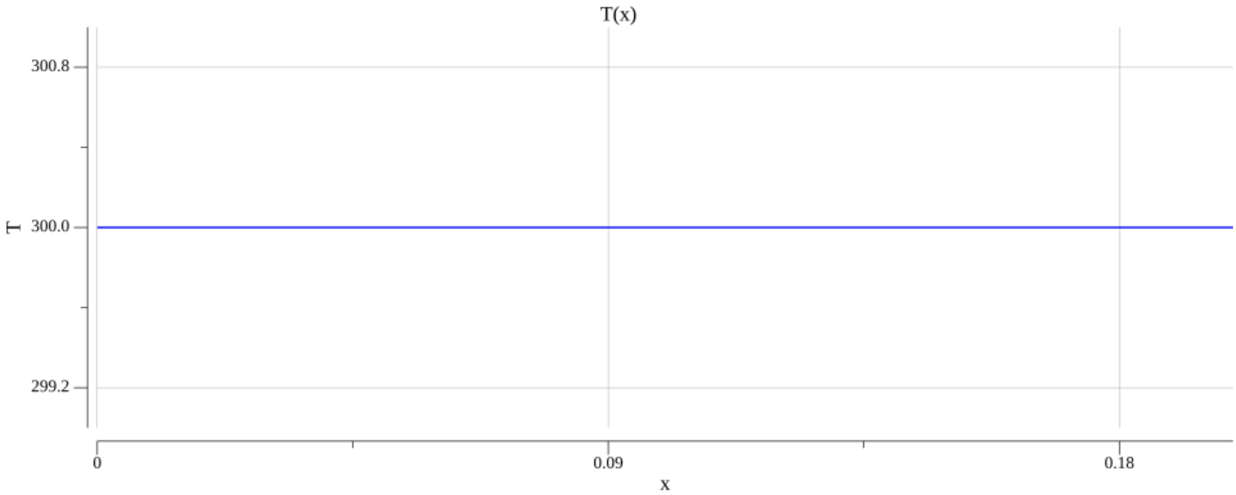

Рисунок 5 – График зависимости $T(x)$ при $F_0 = 0$

Для указанного в задании исходного набора параметров привести данные по балансу энергии.

- $f1 = 27.04$;
- $f1 = 27.01$;
- $\varepsilon_1 = 10^{-5}$ (по температуре);
- $\varepsilon_2 = 10^{-3}$ (по балансу).

Ответы на контрольные вопросы

Вопрос 1. Какие способы тестирования программы можно предложить?

Ответ. В качестве еще одного способа тестирования можно проследить закономерности: при $F_0 > 0$ происходит охлаждение пластины, а при $F_0 < 0$ – нагревание. Кроме того, при увеличении показтеля теплосъема, уровень должен снижаться, а градиент увеличиваться.

Вопрос 2. Получите простейший разностный аналог нелинейного краевого условия при $x = l$.

Ответ. Для получения разностного аналога необходимо изначально аппроксимировать производную:

$$\frac{dT}{dx} = \frac{y_N - y_{N-1}}{h}$$

Подставим полученное выражение в исходное уравнение:

$$-k_N \frac{y_n - y_{N-1}}{h} = \alpha_N(y_N - T_0) + \varphi(y_N)$$

Учтём, что $y_{N-1} = \xi_N y_N + \eta_N$:

$$-k_N(y_N - \xi_N y_N + \eta_N) = \alpha_N(y_N - T_0)h + \varphi(y_N)h$$

Приведя подобные слагаемые, получим:

$$\varphi(y_N)h + (k_N + \alpha_N h - k_N\xi_N - k_N\eta_N)y_N - h\alpha_N T_0 = 0$$

Вопрос 3. Опишите алгоритм применения метода прогонки, если при $x = 0$ краевое условие квазилинейное (как в настоящей работе), а при $x = l$, как в п. 2.

Ответ. Для начала найдем начальные прогочные коэффициенты по формулам (коэффициенты M_0, P_0, K_0 известны):

$$\xi = \frac{-M_0}{P_0}$$

$$\eta = \frac{-K_0}{P_0}$$

Далее, находим последующие прогоночные коэффициенты:

$$\xi_{n+1} = \frac{C_n}{B_n - A_n\xi_n}$$

$$\eta_{n+1} = \frac{F_n + A_n\eta_n}{B_n - A_n\xi_n}$$

Из уравнения, полученного в п. 2., можем получить y_N. По прогочной формуле можем найти все значения неизвестных y_N

$$y_n = \xi_{n+1}y_{n+1} + \eta_{n+1}$$

Вопрос 4. Опишите алгоритм определения единственного значения сеточной функции y_p в одной заданной точке p. Использовать встречную прогонку, т.е. комбинацию правой и левой прогонок.

Ответ. Вычислим начальные прогоночные коэффициенты.

Для правой прогонки:

$$\xi = \frac{-M_0}{P_0}$$

$$\xi = \frac{-K_0}{P_0}$$

Для левой прогонки:

$$\alpha_{N-1} = \frac{-M_N}{K_N}$$

$$\beta_{N-1} = \frac{-P_N}{K_N}$$

Найдем прогоночные коэффициенты.

Для левой прогонки:

$$\xi_{n+1} = \frac{C_n}{B_n - A_n\xi_n}$$

$$\eta_{n+1} = \frac{F_n + A_n\eta_n}{B_n - A_n\xi_n}$$

Для правой прогонки:

$$\alpha_{n-1} = \frac{A_n}{B_n - C_n\alpha_n}$$

$$\beta_{n-1} = \frac{F_n + C_n\beta_n}{B_n - C_n\alpha_n}$$

Левые и правые прогонки:

$$y_n = \xi_{n+1}y_{n+1} + \eta_{n+1}$$

$$y_n = \alpha_{n-1}y_{n+1} + \beta_{n-1}$$

Выразим y_p:

$$y_{p-1} = \xi_p y_p + \eta_p$$

$$y_p = \alpha_{p-1}y_{p-1} + \beta_{p-1}$$

$$y_p = \frac{\xi_{n+1}\beta_n + \eta_{n+1}}{1 - \xi_{n+1}\alpha_n}$$